Homework 3

Saheedat Olasumbo Abbas

2025-10-26

#Smoothing by bin means (bin depth = 3)

```
age <- c(13, 15, 16, 16, 19, 20, 20, 21, 22, 22, 25, 25, 25, 25, 30,
33, 33, 35, 35, 35, 35, 36, 40, 45, 46, 52, 70)
length(age)
```

```
## [1] 27
```

```
summary(age)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   13.00   20.50   25.00   29.96   35.00   70.00
```

```
bin_depth <- 3
stopifnot(length(age) %% bin_depth == 0)
n_bins <- length(age) / bin_depth

# Indices for bins

bins <- split(seq_along(age), rep(1:n_bins, each = bin_depth))

# Compute bin means and smoothed vector

bin_means <- sapply(bins, function(idx) mean(age[idx]))
smoothed <- unlist(mapply(function(idx, m) rep(m, length(idx)), bins, bin_means))

# Show original vs smoothed (first few bins for illustration)

data.frame(
idx = 1:length(age),
original = age,
bin = rep(1:n_bins, each = bin_depth),
bin_mean = rep(round(bin_means, 3), each = bin_depth),
smoothed = round(smoothed, 3)
)
```

```
##    idx original bin bin_mean smoothed.1 smoothed.2 smoothed.3 smoothed.4
## 1    1       13   1   14.667     14.667     18.333         21         24
## 2    2       15   1   14.667     14.667     18.333         21         24
## 3    3       16   1   14.667     14.667     18.333         21         24
## 4    4       16   2   18.333     14.667     18.333         21         24
## 5    5       19   2   18.333     14.667     18.333         21         24
## 6    6       20   2   18.333     14.667     18.333         21         24
## 7    7       20   3   21.000     14.667     18.333         21         24
## 8    8       21   3   21.000     14.667     18.333         21         24
## 9    9       22   3   21.000     14.667     18.333         21         24
## 10  10       22   4   24.000     14.667     18.333         21         24
## 11  11       25   4   24.000     14.667     18.333         21         24
## 12  12       25   4   24.000     14.667     18.333         21         24
## 13  13       25   5   26.667     14.667     18.333         21         24
## 14  14       25   5   26.667     14.667     18.333         21         24
## 15  15       30   5   26.667     14.667     18.333         21         24
## 16  16       33   6   33.667     14.667     18.333         21         24
## 17  17       33   6   33.667     14.667     18.333         21         24
## 18  18       35   6   33.667     14.667     18.333         21         24
## 19  19       35   7   35.000     14.667     18.333         21         24
## 20  20       35   7   35.000     14.667     18.333         21         24
## 21  21       35   7   35.000     14.667     18.333         21         24
## 22  22       36   8   40.333     14.667     18.333         21         24
## 23  23       40   8   40.333     14.667     18.333         21         24
## 24  24       45   8   40.333     14.667     18.333         21         24
## 25  25       46   9   56.000     14.667     18.333         21         24
## 26  26       52   9   56.000     14.667     18.333         21         24
## 27  27       70   9   56.000     14.667     18.333         21         24
##    smoothed.5 smoothed.6 smoothed.7 smoothed.8 smoothed.9
## 1      26.667     33.667         35     40.333         56
## 2      26.667     33.667         35     40.333         56
## 3      26.667     33.667         35     40.333         56
## 4      26.667     33.667         35     40.333         56
## 5      26.667     33.667         35     40.333         56
## 6      26.667     33.667         35     40.333         56
## 7      26.667     33.667         35     40.333         56
## 8      26.667     33.667         35     40.333         56
## 9      26.667     33.667         35     40.333         56
## 10     26.667     33.667         35     40.333         56
## 11     26.667     33.667         35     40.333         56
## 12     26.667     33.667         35     40.333         56
## 13     26.667     33.667         35     40.333         56
## 14     26.667     33.667         35     40.333         56
## 15     26.667     33.667         35     40.333         56
## 16     26.667     33.667         35     40.333         56
## 17     26.667     33.667         35     40.333         56
## 18     26.667     33.667         35     40.333         56
## 19     26.667     33.667         35     40.333         56
## 20     26.667     33.667         35     40.333         56
## 21     26.667     33.667         35     40.333         56
## 22     26.667     33.667         35     40.333         56
## 23     26.667     33.667         35     40.333         56
```

```
## 24     26.667     33.667          35     40.333          56
## 25     26.667     33.667          35     40.333          56
## 26     26.667     33.667          35     40.333          56
## 27     26.667     33.667          35     40.333          56
```

#Outliers via the IQR rule

```
Q1 <- quantile(age, 0.25)
Q3 <- quantile(age, 0.75)
IQR_val <- IQR(age)
lower_fence <- Q1 - 1.5 * IQR_val
upper_fence <- Q3 + 1.5 * IQR_val
outliers <- age[age < lower_fence | age > upper_fence]

list(Q1 = Q1, Q3 = Q3, IQR = IQR_val,
lower_fence = lower_fence, upper_fence = upper_fence,
outliers = outliers)
```

```
## $Q1
##  25%
## 20.5
##
## $Q3
## 75%
##  35
##
## $IQR
## [1] 14.5
##
## $lower_fence
##   25%
## -1.25
##
## $upper_fence
##   75%
## 56.75
##
## $outliers
## [1] 70
```

#Min–max normalization of age = 35 to [0, 1]

```
x <- 35
min_age <- min(age); max_age <- max(age)
minmax_0_1 <- (x - min_age) / (max_age - min_age)
minmax_0_1
```

```
## [1] 0.3859649
```

#Z-score normalization of age = 35

```
mu <- mean(age)
sigma <- sd(age)
z_35 <- (x - mu) / sigma
c(mean = mu, sd = sigma, z_35 = z_35)
```

```
##       mean         sd       z_35
## 29.9629630 12.9421241  0.3891971
```

#Decimal scaling normalization of age = 35

```
j <- ceiling(log10(max(abs(age))))
decimal_35 <- x / (10^j)
c(j = j, decimal_scaled_35 = decimal_35)
```

```
##                 j decimal_scaled_35
##              2.00              0.35
```

#Question 2: Function for Min–Max Normalization to an Arbitrary Range

```
foo <- function(a, min_new, max_new) {
stopifnot(is.numeric(a), is.numeric(min_new), is.numeric(max_new))
if (length(a) == 0) return(numeric())
a_min <- min(a, na.rm = TRUE)
a_max <- max(a, na.rm = TRUE)
if (a_max == a_min) {
# All values identical → map them to the midpoint of the new range
return(rep((min_new + max_new)/2, length(a)))
}
( (a - a_min) / (a_max - a_min) ) * (max_new - min_new) + min_new
}

# Example

foo_0_1 <- foo(age, 0, 1)
foo_10_20 <- foo(age, 10, 20)
head(data.frame(age, to_0_1 = round(foo_0_1, 4), to_10_20 = round(foo_10_20, 2)))
```

```
##   age to_0_1 to_10_20
## 1  13 0.0000    10.00
## 2  15 0.0351    10.35
## 3  16 0.0526    10.53
## 4  16 0.0526    10.53
## 5  19 0.1053    11.05
## 6  20 0.1228    11.23
```

#Q3: Information Gain

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(tibble)

# Data (from the table)
df <- tribble(
  ~department, ~age,    ~salary,    ~status, ~count,
  "sales",     "31_35", "46K_50K",  "senior", 30,
  "sales",     "26_30", "26K_30K",  "junior", 40,
  "sales",     "31_35", "31K_35K",  "junior", 40,
  "systems",   "21_25", "46K_50K",  "junior", 20,
  "systems",   "31_35", "66K_70K",  "senior", 5,
  "systems",   "26_30", "46K_50K",  "junior", 3,
  "systems",   "41_45", "66K_70K",  "senior", 3,
  "marketing", "36_40", "46K_50K",  "senior", 10,
  "marketing", "31_35", "41K_45K",  "junior", 4,
  "secretary", "46_50", "36K_40K",  "senior", 4,
  "secretary", "26_30", "26K_30K",  "junior", 6
)

entropy <- function(counts) {
  p <- counts / sum(counts)
  p <- p[p > 0]
  -sum(p * log2(p))
}

split_entropy <- function(data, attr) {
  groups <- split(data, data[[attr]])
  total <- sum(data$count)
  sum(sapply(groups, function(g) {
    w <- sum(g$count) / total
    class_counts <- tapply(g$count, g$status, sum)
    w * entropy(class_counts)
  }))
}

info_gain <- function(data, attr) {
  overall_counts <- tapply(data$count, data$status, sum)
  H <- entropy(overall_counts)
  H - split_entropy(data, attr)
}

#Root selection
attrs <- c("department", "age", "salary")
ig_root <- sapply(attrs, function(a) info_gain(df, a))
best_root <- names(which.max(ig_root))

cat("Root attribute:", best_root, "\n")
```

```
## Root attribute: salary
```

```r
print(round(ig_root, 5))
```

```
## department        age     salary
##    0.04861    0.42474    0.53752
```

```r
# Second level per branch
remaining_attrs <- setdiff(attrs, best_root)

best_child_split <- function(data, root_attr, branch_value, remaining_attrs) {
  subset <- data %>% filter(.data[[root_attr]] == branch_value)
  class_counts <- tapply(subset$count, subset$status, sum)
  if (length(class_counts[class_counts > 0]) <= 1) {
    return(list(pure = TRUE, best_attr = NA, igs = setNames(numeric(0), character(0))))
  }
  igs <- sapply(remaining_attrs, function(a) info_gain(subset, a))
  list(pure = FALSE, best_attr = names(which.max(igs)), igs = igs)
}

branches <- unique(df[[best_root]])
second_level <- lapply(branches, function(bv) {
  list(branch_value = bv,
       result = best_child_split(df, best_root, bv, remaining_attrs))
})

cat("\nSecond-level choices per", best_root, "branch:\n")
```

```
##
## Second-level choices per salary branch:
```

```r
for (s in second_level) {
  bv <- s$branch_value
  res <- s$result
  cat("\n- Branch:", best_root, "=", bv, "\n")
  if (isTRUE(res$pure)) {
    cat("  Node is pure (no split needed)\n")
  } else {
    cat("  Best second-level attribute:", res$best_attr, "\n")
    print(round(res$igs, 5))
  }
}
```

```
##
## - Branch: salary = 46K_50K
##   Best second-level attribute: department
## department        age
##    0.94682    0.94682
##
## - Branch: salary = 26K_30K
##   Node is pure (no split needed)
##
## - Branch: salary = 31K_35K
##   Node is pure (no split needed)
##
## - Branch: salary = 66K_70K
##   Node is pure (no split needed)
##
## - Branch: salary = 41K_45K
##   Node is pure (no split needed)
##
## - Branch: salary = 36K_40K
##   Node is pure (no split needed)
```

#Question4: If–Then Rules Derived from the Two-Level Tree

```r
build_rules <- function(data, root_attr, second_level_info) {
  rules <- list()
  for (s in second_level_info) {
    bv <- s$branch_value
    subset_root <- data %>% filter(.data[[root_attr]] == bv)

    if (isTRUE(s$result$pure)) {
      counts <- tapply(subset_root$count, subset_root$status, sum)
      majority <- names(which.max(counts))
      rules[[length(rules) + 1]] <- list(
        conditions = setNames(list(bv), root_attr),
        class = majority
      )
    } else {
      child_attr <- s$result$best_attr
      child_vals <- unique(subset_root[[child_attr]])
      for (cv in child_vals) {
        subset_child <- subset_root %>% filter(.data[[child_attr]] == cv)
        if (nrow(subset_child) == 0) next
        counts <- tapply(subset_child$count, subset_child$status, sum)
        counts[is.na(counts)] <- 0
        if (sum(counts) == 0) next
        majority <- names(which.max(counts))
        rules[[length(rules) + 1]] <- list(
          conditions = c(setNames(list(bv), root_attr),
                         setNames(list(cv), child_attr)),
          class = majority
        )
      }
    }
  }
  rules
}

rules <- build_rules(df, best_root, second_level)

print_rule <- function(r) {
  conds <- paste0(names(r$conditions), " = ", unlist(r$conditions), collapse = " AND ")
  paste0("IF ", conds, " THEN status = ", r$class)
}

cat("\nDerived two-level rules:\n")
```

```
##
## Derived two-level rules:
```

```r
cat(paste0("* ", vapply(rules, print_rule, character(1)), collapse = "\n"))
```

```
## * IF salary = 46K_50K AND department = sales THEN status = senior
## * IF salary = 46K_50K AND department = systems THEN status = junior
## * IF salary = 46K_50K AND department = marketing THEN status = senior
## * IF salary = 26K_30K THEN status = junior
## * IF salary = 31K_35K THEN status = junior
## * IF salary = 66K_70K THEN status = senior
## * IF salary = 41K_45K THEN status = junior
## * IF salary = 36K_40K THEN status = senior
```

```
cat("\n")
```